Въведение в алгоритмите и асимптотичния анализ

Тодор Дуков

Що е то алгоритъм?

Алгоритмите се срещат навсякъде около нас:

- рецептите са алгоритми за готвене
- сутрешното приготвяне
- придвижването от точка A до точка B
- търсенето на книга в библиотеката

Въпреки това е трудно да се даде формална дефиниция на това какво точно е алгоритъм. На ниво интуиция, човек може да си мисли, че това просто е някакъв последователен списък от стъпки/инструкции, които човек/машина трябва да изпълни. Други начини човек да си мисли за алгоритмите, са:

- програми – обикновено така се реализират алгоритми
- машини на Тюринг, крайни (стекови) автомати или формални граматики
- частично рекурсивни функции

Един програмист в ежедневието си постоянно пише алгоритми за да решава различни задачи/проблеми. Една задача може да се решава по много начини, някои по-добри от други. Добрият програмист, освен че ще намери решение на проблема, той ще намери най-доброто решение (или поне достатъчно добро за неговите цели).

Какво означава добро решение?

Хубаво е човек да се води по следните (неизчерпателни) критерии:

- решението трябва да е коректно – ако алгоритъма работи само през 50% от времето, най-вероятно можем да се справим по-добре
- решението трябва да е бързо – ако алгоритъма ще завърши работа след като всички звезди са измрели, то той практически не ни върши работа
- решението трябва да заема малко памет – ако алгоритъма по-време на своята работа се нуждае от повече памет, колкото компютъра може да предостави, за наз този алгоритъм е безполезен
- решението трябва да е просто – това е може би най-маловажния критерии от тези, но въпреки това е хубаво когато човек може, да пише чист и разбираем код, който лесно се разширява

За да можем да сравняваме алгоритми в зависимост от това колко големи ресурси (време и памет) използват, трябва първо да можем да квантифицираме тези ресурси.

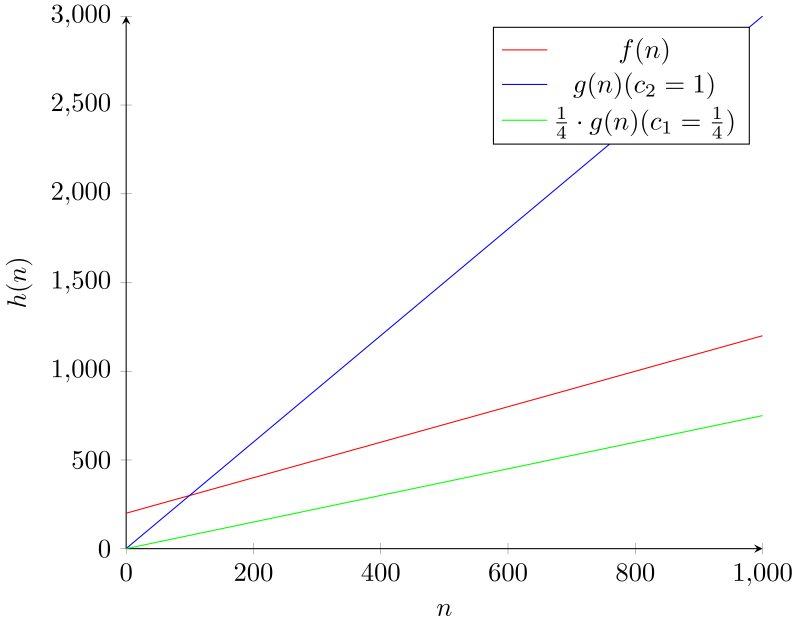
Как квантифицираме времето и паметта?

Когато пишем алгоритми, имаме няколко базови инструкции (за които предварително сме се уговорили), които ще наричаме **атомарни инструкции**. Тяхното извикване ще отнеме една единица време. **Време за изпълнение** ще наричаме броят на извикванията на атомарните инструкции по време на изпълнение на програмата. Също така числата и символите ще бъдат нашите **атомарни типове данни**, и ще заемат една единица памет. **Паметта**, която една програма заема, ще наричаме максималния брой на единици от атомарни типове данни по-време на изпълнение, без да броим входните данни. Обикновено времето и паметта зависят от размера на подадените входни данни. Това означава, че можем да си мислим за времето и паметта като функции на размера на входа. Подхода, който ще изберем, е да сравняваме функциите за време/памет на различните алгоритми асимптотично. Интересуваме се не толкова от конкретните стойности, а от поведението им когато вървим към безкрайността.

Няколко дефиниции

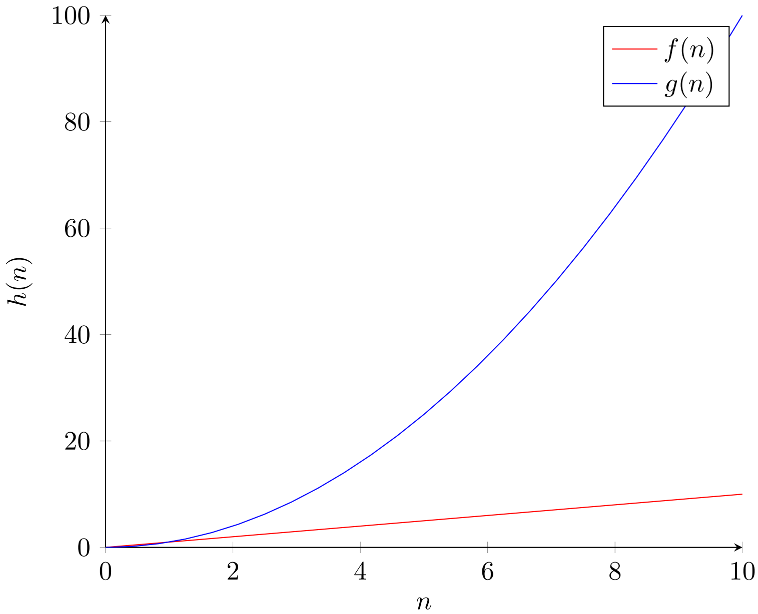
Множеството от функции, които ще анализираме е $\mathcal{F} = \{f \mid f : \mathbb{N} \to \mathbb{R}^{\geq 0}\}$. За всяка функция $f \in \mathcal{F}$ ще дефинираме следните пет множества:

- $\Theta(f) = \{g \in \mathcal{F} \mid (\exists c_1 > 0)(\exists c_2 > 0)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n))\}$
  Може да тълкуваме $\Theta(f)$ като множеството от функциите, които растат* толкова бързо, колкото f. Можем да вземем за пример $f(n) = n + 200$ и $g(n) = 3n + 1$:



  На картинката се вижда как от един момент нататък, функцията f остава "заключена" между $c_1 \cdot g$ и $c_2 \cdot g$. Вместо да пишем $g \in \Theta(f)$, ще пишем $g = \Theta(f)$ или $g \asymp f$.

- $O(f) = \{g \in \mathcal{F} \mid (\exists c > 0)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(f(n) \leq c \cdot g(n))\}$
  Може да тълкуваме $O(f)$ като множеството от функциите, които не растат* по-бързо от f. Тук заслабваме условията от $\Theta(f)$ като искаме само горната граница. Нека вземем за пример $f(n) = n$ и $g(n) = n^2$:



  Вместо да пишем $g \in O(f)$, ще пишем $g = O(f)$ или $g \preceq f$.

- $o(f) = \{g \in \mathcal{F} \mid (\forall c > 0)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(f(n) < c \cdot g(n))\}$
  Може да тълкуваме $o(f)$ като множеството от функциите, които растат* по-бавно от f. Разликата между $O(f)$ и $o(f)$ е строгото неравенство. Лесно се вижда, че $o(f) \subseteq O(f)$. Тук изключваме функциите от същия порядък. Вместо да пишем $g \in o(f)$, ще пишем $g = o(f)$ или $g \prec f$.

*точност до константен множител

- $\Omega(f) = \{g \in \mathcal{F} \mid (\exists c > 0)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(c \cdot g(n) \leq f(n))\}$
  Може да тълкуваме $\Omega(f)$ като множеството от функциите, които не растат* по-бавно от f. Това е дуалното множество на $O(f)$. Вместо да пишем $g \in \Omega(f)$, ще пишем $g = \Omega(f)$ или $g \succeq f$.

- $\omega(f) = \{g \in \mathcal{F} \mid (\forall c > 0)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(c \cdot g(n) < f(n))\}$
  Може да тълкуваме $\omega(f)$ като множеството от функциите, които растат* по-бързо от f. Това е дуалното множество на $o(f)$. Вместо да пишем $g \in \omega(f)$, ще пишем $g = \omega(f)$ или $g \succ f$.

Няколко полезни свойства

Тук ще видим няколко свойства, които много често се ползват в задачите:

- Нека $f, g \in \mathcal{F}$ и $\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = l$ Тогава ако $l = 0$, то $f \prec g$, ако $l = \infty$, то $f \succ g$, и в противен случай $f \asymp g$
- $f + g \asymp \max\{f, g\}$ за всяко $f, g \in \mathcal{F}$
- $c \cdot f \asymp f$ за всяко $f \in F$ и $c > 0$
- $f \asymp g \iff f^c \asymp g^c$ за всяко $f, g \in F$ и $c > 0$
- $O(f) \cap \Omega(f) = \Theta(f)$ за всяко $f \in \mathcal{F}$
- $o(f) \cap \omega(f) = O(f) \cap \omega(f) = o(f) \cap \Omega(f) = \varnothing$ за всяко $f \in \mathcal{F}$
- $f \prec g \iff g \succ f$ и $f \preceq g \iff g \succeq f$ за всяко $f, g \in \mathcal{F}$
- ако $f \prec g$, то $c^f \prec c^g$ за всяко $f, g \in \mathcal{F}$ и $c > 1$
- ако $\log(f) \prec \log(g)$, то $f \prec g$ за всяко $f, g \in \mathcal{F}$ и $c > 1$
- ако $c^f \asymp c^g$, то $f \asymp g$ за всяко $f, g \in \mathcal{F}$ и $c > 1$
- ако $f \asymp g$, то $\log(f) \asymp \log(g)$ за всяко $f, g \in \mathcal{F}$ и $c > 1$
- $n! \asymp \sqrt{n}\frac{n^n}{e^n}$ - апроксимация на Stirling
- $\log(n!) \asymp n\log(n)$
- $\log(n) \prec n^k \prec 2^n \prec n! \prec n^n \prec 2^{n^2}$ за всяко $k \geq 1$

Задачи

Задача 1. Да се сравнят асимптотично следните двойки функции:

1. $f(n) = \log(\log(n))$ и $g(n) = \log(n)$
2. $f(n) = 5n^3$ и $g(n) = n\sqrt{n^9 + n^5}$
3. $f(n) = n5^n$ и $g(n) = n^2 3^n$
4. $f(n) = n^n$ и $g(n) = 3^{n^2}$
5. $f(n) = 3^{n^2}$ и $g(n) = 2^{n^3}$

Задача 2. Да се докаже, че $\sum\limits_{i=0}^{n} i^k \asymp n^{k+1}$

Задача 3. Да се подредят по асимптотично нарастване следните функции:

$f_1(n) = n^2$	$f_2(n) = \sqrt{n}$	$f_3(n) = \log^2(n)$	$f_4(n) = \sqrt{\log(n)!}$
$f_5(n) = \sum\limits_{k=2}^{\log(n)} \dfrac{1}{k}$	$f_6(n) = \log(\log(n))$	$f_7(n) = 2^{2^{\sqrt{n}}}$	$f_8(n) = \dbinom{\binom{n}{3}}{2}$
$f_9(n) = 2^{n^2}$	$f_{10}(n) = 3^{n\sqrt{n}}$	$f_{11}(n) = 2^{\binom{n}{2}}$	$f_{12}(n) = \sum\limits_{k=1}^{n^2} \dfrac{1}{2^k}$